WOSSS19

Report on the Workshop on Sustainable Software Sustainability 2019 (WOSSS19)

Data Archiving and Networked Services DANS

Software Sustainability Institute

netherlands eScience center

A workshop organised by:
Data Archiving and Networked Services (DANS)
The Software Sustainability Institute (SSI)
The Netherlands eScience Centre

24-26 April 2019
The Hague, the Netherlands

Report editors:

Shoaib Sufi	University of Manchester
Carlos Martinez	Netherlands eScience Center
Cees Hof	Data Archiving and Networked Services (DANS)
Patrick Aerts	Data Archiving and Networked Services (DANS) & Netherlands eScience Center

Report authors and workshop contributors (alphabetical by first name):

Adriaan Klinkenberg	Elsevier
Anna-Lena Lamprecht	Utrecht University
Barbara Sierman	KB National Library of the Netherlands
Bettine van Willigen	LifeTec Group
Brett Olivier	VU University Amsterdam
Carlos Martinez	Netherlands eScience Center
Carol Willing	Project Jupyter
Carsten Thiel	CESSDA ERIC
Caspar van Leeuwen	SURFsara
Catherine Jones	STFC
Cees Hof	Data Archiving and Networked Services (DANS)
Christina von Flach	Federal University of Bahia
Daniel S. Katz	University of Illinois at Urbana-Champaign
Dominique Hansen	Humboldt-Universität zu Berlin
Esther Plomp	Delft University of Technology
Gerard Coen	Data Archiving and Networked Services (DANS)
Hamish Steptoe	Met Office, UK
Hannah Bosma	Universiteit van Amsterdam
Heather Andrews	Delft University of Technology
James Davenport	University of Bath
Jamie Shiers	CERN
Jesse de Vos	Beeld en Geluid
Johan van der Knijff	KB National Library of the Netherlands
Jurriaan H. Spaaks	Netherlands eScience Center
Kostas Kavoussanakis	EPCC, University of Edinburgh
Leyla Garcia	ELIXIR Hub
Mario Behn	NATO Communication and Information Agency
Mario David	LIP Lisbon
Mateusz Kuzak	Netherlands eScience Center
Neil Chue Hong	University of Edinburgh
Nicolas Dintzner	Delft University of Technology

Pablo Orviz	CSIC - Spanish National Research Council
Patrick Aerts	Data Archiving and Networked Services (DANS) & Netherlands eScience Center
Paula Martinez Lavanchy	Delft University of Technology - Library
Peter Doorn	Data Archiving and Networked Services (DANS)
Rachael Kotarski	British Library
Raniere Silva	University of Manchester
Robert Haines	University of Manchester
Roberto di Cosmo	Software Heritage Archive
Shoaib Sufi	University of Manchester
Skip Overgoor	VU University Amsterdam
Stephan Druskat	Friedrich Schiller University Jena & German Aerospace Centre (DLR)
Stephanie van de Sandt	CERN
Tim van den Boom	LifeTec Group
Viviana Letizia	SoftwareX, Elsevier
Wiel Seuskens	LIMA
Yoann Moranville	DARIAH

Table of Contents

1 Introduction

This report is based on the discussions and presentations that took place at the Workshop on Sustainable Software Sustainability[1] in April 2019 in The Hague (WOSSS19). It captures the state of the art for a range of software sustainability themes that were brought up by the organisers and attendees of the workshop.

1.1 How to cite this report

This Digital Object Identifier (DOI[2]) for this report is 10.5281/zenodo.3922155 – it can be resolved by visiting https://www.doi.org/. The report is hosted on Zenodo[3], please see the 'Share Cite as' section of the resolved page to formulate a citation in the appropriate citation style needed.

[1] https://www.software.ac.uk/wosss19

[2] https://en.wikipedia.org/wiki/Digital_object_identifier

[3] https://zenodo.org

2 About the organisers

WOSSS19 was organised by Data Archiving and Networked Services (DANS), an institute of The Royal Netherlands Academy of Arts and Sciences (KNAW), The UK Software Sustainability Institute (SSI) and the Netherlands eScience Center.

2.1 About DANS

DANS[1] is the Netherlands Institute for permanent access to digital research resources, DANS encourages researchers to make their digital research data and related outputs Findable, Accessible, Interoperable and Reusable (FAIR). To support the research community three technical core services are offered: DataverseNL for short-term data management, EASY for long-term archiving, and NARCIS, the national portal for research information. Next to these technical core services DANS also provides expert advice, training and certified services. By participating in (inter)national projects, networks and research, DANS contributes to continued innovation of the global scientific data infrastructure. DANS is funded by the Royal Netherlands Academy of Arts and Sciences (KNAW) and the Dutch national scientific funding organisation NWO.

2.2 About SSI

The Software Sustainability Institute[2] - SSI(1) (www.software.ac.uk) is a leading international authority on research software sustainability, working with researchers, funders, software engineers, managers, and other stakeholders across the research spectrum. The use of software in research continues to grow, and the Institute plays a key role in helping the research community to help itself. The SSI have built a network of over 140 Fellows (www.software.ac.uk/fellows) from across research disciplines, championed research software and software career paths to stakeholders, worked with over 70 projects to improve their codes (www.software.ac.uk/consultancy/consultancy-testimonials), written guides (www.software.ac.uk/resources/guides) on all aspects of software sustainability read by over 50,000 people, and organised training events for thousands of learners. The SSI mission is to cultivate better, more sustainable, research software to enable world-class research ("Better Software, Better Research").

2.3 About the Netherlands eScience Center

The Netherlands eScience Center[3] is the Dutch center of excellence for research software. It collaborates with both academia and industry, on the interface of e-infrastructure (computing, data), data science and domain sciences varying from climate science, astronomy, chemistry to humanities amongst others. The eScience Center has expertise in data handling, big data analytics and efficient computing. It maintains an eScience technology platform containing tools, interfaces, and libraries to deal with and extract information from large amounts of (distributed) data, requiring large computing infrastructures, high-speed networks, and high-resolution visualisation equipment.

[1] https://dans.knaw.nl/en

[2] http://www.software.ac.uk

[3] http://www.esciencecenter.nl

The eScience Center aims to improve the quality and sustainability of research software by promoting best practices in scientific software development and developing the skills of scientific software developers.

3 Method

Key attendees introduced a particular topic in the workshop, highlighting the conversation in that space. Topics and their subtopics were then discussed further. The sections in this report follow this ‘introduction’ and ‘discussion’ structure.

There were three main themes at the workshop: preventing new legacy, legacy & recovery and policy and quality. Each theme had a number of subtopics and members standing in the community of research software, software preservation and software policy were invited to give an introduction to those areas.

The table below outlines the topics and their subtopics that were introduced:

Topic	Subtopics
Preventing new legacy	• Software development • FAIR Principles for software • EOSC, FAIR and Software • Software deposition routes • Software Directories • Software Heritage Archive • Citation and rewards systems • Containers • Resourcing and planning
Legacy and recovery	• Legacy issues • Heritage matters • Cultural sector issues • Recovery issues and efforts • Re-use metrics
Policy and quality	• Quality metrics • A European Software Sustainability Infrastructure • The European Open Science Cloud (EOSC)

After introductions on a topic, attendees had a choice to join a discussion on one of the subtopics. These groups started by discussing matters of relevance to the subtopic such as evidenced practice, currently considered best practices, current problems and future solutions in that space. These larger groups then broke into smaller subgroups which discussed certain aspects of the subtopic further. After discussions, these subgroups then worked together to produce some initial draft text. The smaller subgroups used a co-production method pioneered by the use of collaborative writing called speed blogging (www.software.ac.uk/speed-blogging-and-tips-writing-speed-blog-post).

After the workshop, the texts produced in the subgroups were then organised into sections and revised by the WOSSS19 editorial group. As all attendees are counted as authors, they were then given time to provide comments and add any additional clarifying points to the sections. Those who had provided introductions to subtopics were given the opportunity to update or write their introductory sections to match the slides they presented.

After this stage, one of the editors went through the document incorporating suggested changes, applying consistent formatting and standardising references so that the report was ready for publication.

4 Executive summary and key recommendations

4.1 Summary

The analysis of the workshop outputs led to specific recommendations in six key areas: credit, cultural heritage, FAIR, funding, policy and training.

There is essential work needed to allow credit for software to become more specific and traceable in research. The Cultural Heritage sector needs to be more aware of the implications of their technology choices. The FAIRification of software has implications for the work on funders tracking research outputs and software journals. Funders should prioritise software maintenance, support the work of credit for software and there should be funded coordination of software sustainability efforts in Europe. Policy initiatives should reflect that supporting sustainability is an ongoing effort and further coordinated work is needed on career paths as well as software directories. Training efforts should be available for all levels of practice and the methods of the Carpentries initiative should be employed to help scale production and delivery of training.

4.2 Key recommendations

4.2.1 Credit

- Stakeholders (e.g. publishers, institutions, funders, researchers, research software engineers) need to agree on how credit should be attributed and valued for the various roles involved in producing research software (e.g. principal investigators, architects, designers, researchers, developers).
- The Software Citation community needs to include mechanisms to attribute how dependent research is on particular software.
- National facilities, institutions and funders should support the creation and running of software directories to highlight the impact of software produced in their areas of work and encourage reuse and credit for those involved in the production of the software.
- Stakeholders (e.g. publishers, libraries) need to support the inclusion of software in the citation graph of research.

4.2.2 Cultural Heritage

- Cultural Heritage organisations need to actively manage the risk of using commercial cloud infrastructures. As preserving world heritage and commercial concerns tend to work on different timescales.

4.2.3 FAIR

- Publishers should look to the efforts around FAIRification of research software to devise the metadata requirements of software journal publications.
- Funders should mandate software is included alongside data in Output Management Plans (i.e. Data Management Plans should move beyond data only and FAIRification should be applied to other project outputs).

4.2.4 Funding

- Funders should fund the maintenance of code and the robustness of associated services once the novelty of functionality and impact have been demonstrated.
- Institutions need to become more aware of the impact of their software contributions and thus make strategic investment in the maintenance and development of key software.
- Funders need to support the coordination of technical and social challenges in the software citation space to maintain the momentum of this key activity.
- Funders and governments need to invest in national Sustainable Software initiatives where there is currently no provision and continued national investment is required where there is.
- Funders and related stakeholders across Europe should support the creation of a European Software Sustainability Infrastructure to work across national and regional software sustainability efforts, share best practice and level up regions where activity is nascent.

4.2.5 Policy

- Funder, publisher and institutional policies should support software outputs as a first-class citizen of the research process akin to publications and data.
- Funders' policies should reflect that supporting activities (e.g. software sustainability or preservation) are ongoing endeavours and not goals that can be reached.
- Stakeholders (e.g. funders, institutions, research software engineering) should work on establishing career paths and credit for those involved in aiding research software and data's sustainability. This activity is essential if we are to build the long-term human capital needed to support reproducible and reusable research.
- Publishers and support organisations should band together to form a governing body to discuss, share and set principles, standards and best practices in the space of software directories.

4.2.6 Training

- Computational training initiatives must address all levels of software engineering practices from coding to tooling, maintenance and re-use.
- Computational Training initiatives should make use of the resources and methods of the Carpentries (carpentries.org) initiative. This will aid the production of training material, training delivery and instructor capacity building, using a tried and tested approach.

5 Background to the workshop

The organisation of WOSSS19 was inspired by a number of factors:

1. An acknowledgement that so many digitally created documents especially from 1985 to 1999 have become inaccessible. It was the beginning of a new era, in which digitally born material was the new way of working and no one considered that one day, this material would become "legacy". Institutions are facing the problem of recovering their history. How can we learn and help each other to resolve the barriers raised by doing this recovery?

2. What best practices, guidance, education will stop us from being caught in the legacy trap in the future? What planning, initial steps, new ways of working and quality checks should be in place so that long term sustainability of software is baked into the process and not an afterthought?

3. What institutional, national and trans-national policies and structures are needed to keep the conversation around the long-term sustainability of software an active topic in the research and cultural heritage sectors?

WOSSS19 aimed to take stock of the current state in Software Sustainability and make a statement on future needs and activities.

The WOSSS19 workshop followed on from the Sustainable Software Sustainability Workshop (2) that took place in The Hague in 2017 (also known as WOSSS17). WOSSS17 covered topics such as FAIR for software, the cultural sector, infrastructure and the software seal of approval.

WOSSS17 followed on from the previous Knowledge Exchange Workshop on Research Software Sustainability (3) in Berlin in 2015. This workshop was focused on policy matters pertaining to technical and social barriers to sustainable software, the funding landscape and national activities in this space.

The 2017 and 2015 workshops were European-led efforts to help consolidate expertise and move the conversation forward around sustainable software and its importance to research and culture.

The US-based but internationally focused Working towards Sustainable Software for Science: Practices and Experiences (WSSSPE)[1] series of workshops has been an ongoing activity in the software sustainability space since 2013.

Many previous attendees from the WOSSS17 and Knowledge Exchange Workshop were present at the WOSSS19 workshop. In addition, the lead of the WSSSPE initiative was also present at the WOSSS19 workshop.

All of these factors enhanced the topic introductions and discussions by including leading voices in the area of Software Sustainability.

[1] http://wssspe.researchcomputing.org.uk

6 Preventing new Legacy

6.1 Software Development

6.1.1 Introduction

There are established sets of practices in software engineering which are agreed as key contributions to the maintainability of the software. These include version control, coding standards, code review, pair programming, testing, code coverage, continuous integration, refactoring, documentation, semantic versioning, use of standards and being correct before being optimised.

The results of a study (4) involving Research Software Engineers showed that the intrinsic sustainability of the software artefact itself (as focused on in these best practices) was only part of the sustainability story. The extrinsic sustainability of the software was indicated to be just as important. The extrinsic sustainability pertains to the environment in which the software is developed. Examples of extrinsic qualities include how open the software is, whether it's developed by a community versus an individual, how actively maintained it is and whether or not it is resourced.

Software which scores highly on intrinsic factors can be said to be sustainable, however, necessary those factors may be, it is only the addition of the extrinsic factors which allow the software to be in a state which can be termed sustained. Excellent software will not sustain itself.

There are some open questions which follow on from this. Such as what are the most important intrinsic and extrinsic factors - can we prioritise them? How do we get better at the extrinsic factors? When do we stop sustaining software?

6.1.2 Discussions

Code modernisation can cause problems. For example, a piece of code inherited from one's supervisor might give different answers due to a newer compiler being used; the problem here is not doing exactly what was originally done with the code. A 40 year old code developed and used by NATO[1] is a case in point. There are concerns that any refactoring to make the code 'better' might actually lead it to giving incorrect results. Emulation or reverse engineering from binaries can lead to problems around correctness also. How do you know the results and behaviour are the same as they were in the original environment? Code modernisation needs can also arise through changes in practice; e.g. the move from Perl to Python in the 2000's in the Life Sciences. Legacy can thus be caused by changes in usage as well as inactivity in maintenance.

Old codes are not necessarily legacy codes. Coming back to the example of the NATO code - it has over 40 years of contributions, dependencies on external vendors and is used for day to day business. Despite its age it's usage, importance and constant updates keep it from being classified as legacy.

[1] http://www.nato.int

Software metrics can help us understand and prioritise which codes (legacy or not) are worth sustaining or when it’s time to archive them. Minimum standards to class code as legacy would be useful to help decide.

Another problem that can lead to new legacy being made is the use and extension of prototype systems. As an example, Met Office[1] sandbox research prototypes become attractive for operational purposes and get relied on before being operationally supported - this introduces the risk of software tools disappearing without notice. The Met Office thus implemented a risk framework, based on a risk assessment of the likelihood of something going wrong and the impact if something did go wrong:

Likelihood

- What is the complexity of code/science being implemented?
- How experienced are the people writing the code? (Are they new to the language/code development?)
- How dependent is the code on external input data/routines?
- Is this a new code development, or just minor changes to existing code?
- Are there pressing time constraints on the project?

Impact

- Are there customers reliant on this code?
- What are the dependencies of the code? - i.e. will there be implications on other code/systems if there is a problem with this code?
- How 'high-profile' is this project (brand implications if something went wrong)?
- How easy would this code be to fix? Would there be a significant resource requirement?
- Are there any 'single point of failures' associated with the code?
- Is this code part of an internal or external project?

Legacy software, heritage software, hand me down software and use of prototype software need not inherently be a problem; it becomes problematic when usage of these types of software cases catch you out unexpectedly. Thus, understanding the risk posed by these types of software is key to protecting yourself from problems. If you are aware of the problems, you can plan accordingly.

Education in writing software also remains key to researchers, students, research software engineers and even those outside computing/computational disciplines. An improved understanding is needed of how software can be kept manageable and not exhibit all of the bad aspects of what we term legacy.

The Funders are focused on novelty, but where there is a case and the software is seen as enabling for research, they should have systems to fund software productisation and the sustainability of code.

[1] http://www.metoffice.gov.uk

6.2 FAIR Principles for Software

6.2.1 Introduction

FAIR (5) (Findable, Accessible, Interoperable and Reusable) is a move to make research data sharing standards more than just about disclosure.

The question is whether the FAIR principles also work for software. In work that has begun to help specify or guide what FAIR would mean for software, the Top 10 FAIR Data & Software Things (6) is a notable attempt. The Top 10 FAIR Data & Software things advocates for APIs to aid interoperability and reusability.

6.2.1.1 Update

There has been further work planned to move the conversation forward. There were discussions at the deRSE 2019 meeting in June 2019 and the topic was covered at the DTL Communities@Work 2019 workshop in October 2019. There was also to be a session at the March 2020 RDA plenary[1], which led to the creation of the FAIR4RS working group[2].

Work after the WOSSS19 workshop has led to the 'Towards FAIR principles for Research Software' (7), which provides a statement of how FAIR software differs from FAIR data and what additional needs of software are not necessarily covered by the FAIR principles.

6.2.2 Discussions

Sustainable software is the result of the work of the developer(s); sustained software depends on the continued dedication of supported individuals (including developers) to serve, participate and develop communities.

The FAIR (Findable, Accessible, Interoperable, Reusable) principles have gained a lot of attention from the different stakeholders involved in the research process. The FAIR principles are driving institutional data management policies, funder requirements and they have influenced journal data sharing policies. Thus, the term FAIR associated with well-managed software can influence research stakeholders. FAIR is a recognisable term that is now commonly associated with adherence to certain best-practices.

The FAIR principles are about well-documented, open/standard formatted, long-term archived and responsible managed/shared data. The original description of the FAIR principles included software as well as data.

> *Importantly, it is our intent that the principles apply not only to 'data' in the conventional sense, but also to the algorithms, tools, and workflows that led to that data. (5)*

[1] The RDA 2020 was cancelled due to the COVID-19 pandemic - http://www.who.int/emergencies/diseases/novel-coronavirus-2019

[2] https://www.rd-alliance.org/groups/fair-4-research-software-fair4rs-wg

In practice, the FAIR principles have not directly addressed software sustainability. They do however serve as a high-level baseline for what software needs to comply with in order to be sustainable.

FAIR principles are tailored towards preservation (i.e. of data) rather than the broader definition of sustainability which encompasses the ability to continue to use functionality. This is because for software, we consider that the software may evolve as its ecosystem changes, so that the functions that software provides continue whilst the specific versions of software change (and perhaps are preserved). FAIR covers some of the aspects required to help this evolution of software. For instance, it encourages access to the software via persistent identifiers, a clear statement of the license, and sufficient documentation.

It does not, however, guarantee access to the source code, say anything about the community contributing to the sustainability of the software, and cover the testing and "integrity" schedule that might be required to check on a regular basis that the functionality and requirements on the software continue to be provided. It is also clear that, unlike some data, we may not always know how long a piece of software will require to be sustained. It is currently unclear how this might be incorporated into any extension of FAIR.

Since it is still unclear how the FAIR principles for data translate to software, we define what FAIR *should* enable, rather than what it *does* enable. The FAIR principles for software should enable the following in terms of findability, accessibility, interoperability and reusability:

- **F**indable: Search and retrieve based on identifiers and descriptors (meta-data). Examples: functionality, language, software version, where it is deposited.
- **A**ccessible: Allow one to know if they can use the software. Examples: terms of use, licences, membership/registration.
- **I**nteroperable: What kind of environment is needed in order for the software to work. Examples: hardware requirements, operating system, dependencies, input/output formats used.
- **R**eusable: How to use the software? Examples: installation instructions, user manual, developer documentation.

The above is a minimal list that should be feasible to meet by researchers writing software. Describing how to get to this minimum level of acceptable FAIR software is also needed.

Applying the FAIR principles from the beginning of a project will contribute to it eventually achieving sustainability. However, sustainability will be achieved only with the commitment from supported developers to serve their community.

6.3 Software Directories and Deposition Routes

6.3.1 Introduction

The FAIRness of research data is very much determined by the platforms, repositories and directories used to manage, store and disseminate the data. For software the same applies, but with additional complexities around additional metadata and services to support FAIR software.

6.3.1.1 Software Directories

Software Directories allow others to judge the usefulness of software for their needs, enable best practice such as software citation, and allow organisations and funders insights into the impact of the software that they are supporting or funding.

Directories can be domain specific (e.g. bio.tools[1] in the Life Sciences), language specific (e.g. PyPI[2] for the Python programming language), organisational (e.g. KB Lab[3] at the National Library of the Netherlands) or even metalevel directories such as the Science Gateways Communities initiative (SGCI[4]) which focuses on helping people set up gateways and directories.

Integration with existing repositories would decrease submitter workload when envisaging the construction of Software Directories. Integration with systems such as GitHub[5], Zenodo[6] and Zotero[7] are seen as key at the time of writing.

The ability to show the impact of the software via its mentions in publications, use on projects and testimonials by users make for a compelling use case for a Software Directory. The Research Software Directory[8] run by the Netherlands eScience Center aims to be such a directory.

Some open challenges remain such as: how to get people to adopt using directories as part of their workflow? How do we reward and create job profiles for those who curate such directories? How do we incentivise institutes and funders in mandating or encouraging the use of Software Directories?

6.3.1.2 Deposition Routes

The Dutch FAIR Software route is part of a joint project between DANS and the Netherlands eScience Center. The idea is to design a standardised route to deposit software (code and scripts) during or after a research project that will guarantee storage, access, re-use and citation. By providing a route which is easy to use and provides value to the researcher, the idea is to make the deposition of software too easy not to do.

The guiding principles of the route are motivated by open science (accessibility, accountability and sustainability), reproducibility and the FAIR principles (Findability, Accessibility, Interoperability and Reusability). We acknowledge that the FAIR principles for software is an evolving area (see 6.2 on FAIR Principles for Software). Persistence and availability over a 10-year period is a particular goal of directories.

[1] https://bio.tools

[2] https://pypi.org

[3] https://lab.kb.nl

[4] https://sciencegateways.org

[5] https://github.com

[6] https://zenodo.org

[7] https://www.zotero.org

[8] https://research-software.nl

An online platform was developed in summer 2019 as part of the Dutch FAIR Software route. Some of the lessons we learned are that guidance is needed on metadata, having the 'right' kind of persistent identifiers and using the 'right' repository. Different levels of FAIR compliance for different types of software were observed, akin to different curation levels for datasets. There is a balance also between resilience and control. By integrating with external repositories (e.g. Zenodo) you lessen the consequence of failure of the main Directory (thus increasing resilience) but increase the number of points of failure (thus decreasing the control over the whole data record).

Whatever the route becomes, it needs input from those who use, develop and manage it (researchers, developers and curators). It also needs to add value and thus incentivise its use (both internally and externally - i.e. through the benefits it gives users and as an act of compliance to e.g. funder requirements). Training will also need to be provided to make using the system straightforward to users.

6.3.1.3 Update

A joint project between DANS and the Netherlands eScience Center resulted in the creation of the fair-software.eu website. The website, which was officially launched during the National eScience Symposium on 21 November 2019[1], provides recommendations to researchers on ways to improve the quality and reproducibility of their software in the spirit of the FAIR principles.

6.3.2 Discussions

6.3.2.1 Introduction

As the amount of available research software grows, carefully designed and built directories are one method to meet the needs of the diverse set of stakeholders in the software.

Funding, ownership and governance are key facets of software directories. What data and metadata are stored is also a key requirement.

Training and curation need to be factored into the running of such a directory and making use of the lessons learned from current and past directories is essential for the efficient use of resources.

6.3.2.2 Stakeholders:

Multiple stakeholders have an interest in the use of software directories:

- **Funders**: With an interest in identifying software they have funded, and its impact.
- **Institutions**: Who want to identify and promote code created by their employees, encourage use and collaboration between their own staff and track the impact of any institutional funding.

[1] https://www.openaccess.nl/en/events/national-escience-symposium-2019

- **Developers**: Who want the directory to include their code so that it can be found, reused, cited and be used as a reference for career credit.
- (Potential) **Users of code**: Looking for code that they can use to solve their problems or reuse and/or adapt for advanced users.
- **The repository community** (e.g. librarians, publishers, other repositories): Who may wish to enable wider discovery of the software that they're curating.
- **Other directories**: That may wish to aggregate or enable federated search.

Each of these stakeholders will have their own needs which may differ across disciplinary and geographic boundaries. Any directory should design for a core audience, but not in a way that other users are unable to make use of it. These stakeholders break down into two broad groups: those who primarily want to find software and those that want to make it findable.

There's little research currently on how researchers go about finding software that they want to use, whether directories already play a part in this, and what part they specifically play.

Further research here would support the cost-benefit analysis for directories, as well as identify research use cases that need to be supported.

6.3.2.3 Goals of a directory

In order to support the different goals of the various stakeholders, minimally, the directory needs to have the following goals:

- Help users find software.
- Help publishers/producers make software findable.
- Highlight the sustainability of a piece of software.
- Showcase content for a stakeholder (e.g. funders, organisations and producers).

6.3.2.4 Technical requirements

There are technical requirements for software directories to allow them to reach their goal:
- Have a mechanism for getting content into the directory.
- Curation features for those looking after the directory.
- Choose implementation technologies which are themselves likely to be supported long term.
- Ability to export data for future migrations or integration into other projects.

6.3.2.5 Organizational requirements

There are key organisational requirements:
- Consistent medium-term funding with a route to further funding.
- A governance model that covers decision making for the directory and how to take in external contributions to the directory itself.

6.3.2.6 Metadata requirements

There are also metadata requirements:

- The metadata required for each software entry:
    - What will be provided by users; those who wish to find software.
    - What will be provided by developers; those who wish to make it findable.
    - What statistics can be generated or harvested from elsewhere, e.g. Altmetric[1].

6.3.2.7 Community and culture

The success of any initiative depends on a sufficient level of adoption by the community. Mechanisms need to be adopted to allow a degree of co-ownership and participation in developing policy, governance and technical additions by the community.

Deposition levels will be greatly improved when user needs are being met and a quality service is being offered.

Generic software directories can only go so far, if further information is needed then this may exist in domain specific directories. What is essential here is that interoperability (e.g. via APIs) is possible such that aggregation and views on software tailored to user needs are possible.

The Quality of information in the directory is important if it is to gain trust from users. Thus, curation mechanisms, minimum requirements from authors, automated metadata collection, guided submission and assessment of FAIRness and other best practice will render use and longer-term value in a repository.

Journals with their associated characteristics (quality control, impact factor, brand, reputation, sustainable business model) offer clear benefits. How do we get the community as a whole to agree on a new set of principles, develop standards, share best practice? A new body is required to coordinate these actions and improve uptake in the space of software directories.

6.3.2.8 Further discussion on directories

Other relevant discussion on Software Repositories (8) focuses on automating the release of software in a way that updates the relevant repositories, and where the metadata for a release itself has a DOI[2]. Metadata for a release should meet some community-agreed minimum standards with the option of being enriched with domain-specific metadata. The credit associated with the release of software should not only be that of authors but also include funders' contributions. The quality data of the software should include auto-generated results (e.g. statements on which tests the software passed) as well as user contributed testimonials. The ultimate test would be whether directory scores and information could compete or have a similar influence to word of mouth recommendations. A software 'Wikipedia' may encourage collaboration over competition in this space.

[1] https://www.altmetric.com

[2] https://www.doi.org

6.3.2.9 A note on software publishing

In recent years, there has been an emergence of software publishing journals such as JORS[1], JOSS[2], and SoftwareX[3]. These journals do peer review of submitted software packages. The software itself is still hosted on collaborative development sites such as GitHub. The publication itself may keep specific metadata to allow the software to be peer reviewed. This includes the used operating system, programming languages, libraries and other factors. More recently for SoftwareX the software is also automatically executed on CodeOcean[4].

Requiring software to be FAIR could be a beneficial step for software publishing platforms. At the same time the definition of FAIR for software could learn from the requirements used by software journal publishers (9). The research community would benefit from published software being more easily usable.

6.3.3 Open Questions

It remains an open question and area of work as to how to encourage people to adopt best practices in publicising software in a directory.

In addition, integrations with institutional and funder repositories remains a possible route to ease the process of registration of software in the research sphere. Guidelines in this area (e.g. Code4REF[5]) are a nascent and much needed area of activity.

It remains an open question what the minimum requirements are for a software directory. Also, are generic repositories acceptable or is there a need for dedicated domain specific repositories?

Software itself can be complex and there is a need to keep the link between software, data and services intact. Do we need integrated storage and provisioning of computational environments? Or is this going too far and likely to fail and be expensive to do? There are efforts in this space such as the Research Object[6] initiative as a manifestation of FAIR Digital Objects (10).

Software Management in the context of the Research Data Management lifecycle is also an open area for discussion (11): where should software fit in and alongside data? Should other aspects of research also be managed in the more wide ranging scope of Output Management Plans (12)?

[1] https://openresearchsoftware.metajnl.com

[2] https://joss.theoj.org

[3] https://www.journals.elsevier.com/softwarex

[4] https://codeocean.com

[5] https://code4ref.github.io

[6] http://www.researchobject.org

6.4 Citation and reward systems

6.4.1 Introduction

The FORCE11 Software Citation Working[1] group began in July 2015. It was modelled on the work done by the Data Citation group[2]. The group focused on producing a set of software citation principles to encourage adoption across disciplines. By September 2015 it had built a community of researchers, developers, publishers, repositories and librarians. Through lots of engaged discussion, the group arrived at a consensus and published the software citation principles paper in September of 2016 (13).

The paper focused on the following principles:

1. **Importance** - Software's importance to research should be recognised by citation.
2. **Credit and Attribution** - Software should count towards career credit and be attributed to its authors.
3. **Unique Identification** - Software citation should include unique identification.
4. **Persistence** - Software metadata should be available beyond the lifespan of the software.
5. **Accessibility** - Software citation should facilitate its use.
6. **Specificity** - Software citation should be specific to different versions.

The paper was informed by use cases, related works, many discussions, feedback from the community and draft reviews as well as workshops. The paper included lots of discussion on how to use the principles. By early 2017 the group driving the principles declared success!

Shortly thereafter, however, they realised the principles were not enough and a 'small' amount of detail was needed to help implement the principles. So, work began to implement the principles with community involvement from publishers, at conferences, repositories, indexers, funders and others. This led to the start of the FORCE11 Software Citation Implementation Working Group[3].

The group represented a diverse set of interests and expertise and worked on a number of activities including: metadata standards and translation between them (e.g. DataCite Schema 4.1 (14), CodeMeta[4] and Citation File Format (CFF[5]) files)), open source archiving and identification (Software Heritage), and engagement from members of the astronomy, Earth science, mathematics, and high energy physics research communities.

[1] https://www.force11.org/group/software-citation-working-group

[2] https://www.force11.org/group/data-citation-synthesis-working-group

[3] https://www.force11.org/group/software-citation-implementation-working-group

[4] https://codemeta.github.io

[5] https://citation-file-format.github.io

Earth science examples included the ESIP Software and Services Citation Cluster[1] which was strongly aligned with the FORCE11 working group and represented an effective balance of principles and practices. The Enabling FAIR Data project[2] also considered other research outputs such as software.

As the work progressed it was realised that the 'small amount' of extra detail was an underestimation of the work that is still needed. The work of the group represents scattered progress and underlying challenges remained unaddressed (15), although their documentation has aided the direction of further discussions.

Technical complexity and challenges remain. Software has many facets, e.g., license, availability, and approach to versioning. Is determining the correct citation metadata the right of the software authors or can others (e.g. software depositors, librarians) also be responsible for specifying this? The concept of software authorship is also not yet conclusively defined. Unique and uniform identification of software by these facets is being looked into by a new working group, the Joint FORCE11/RDA Software Source Code Identification Working Group[3]. The focus of this group is on building concrete and compatible recommendations for the academic community on how to define and store, access and convert and track citations across software versions.

The social challenges remain daunting. There is a need for groups that work on implementations in context to work together to run pilots and establish norms. Disciplinary communities, publishers, repositories, indexers, funders and institutions need to work together. Creating the reward and motivational structures to allow this to happen is challenging.

There have been many positive developments in 2019 in this space. These include the Software Citation Checklist for Authors (16), the Software Citation Checklist for Developers (17), the work on Software Citation Checklist for Reviewers and CodeMeta's alignment with the schema.org web metadata initiative. However continued work has brought up technical, resource and efficacy challenges. Will the work be useful, used, be completed or is this work a steppingstone which will allow future iterations to take place and thus necessary but understandably not sufficient for the challenges of software citation?

Getting credit for software in and across fields, measuring its impact, identification of software and storing all necessary metadata for citation remain some key challenges in this area of work.

The need for realistic scope and funded work to address the coordination, technical and social challenges in the software citation space is clear. Support is vital if we are to establish clarity and unity in this important area of getting credit and awareness of software in research.

[1] http://wiki.esipfed.org/index.php/Software_and_Services_Citations

[2] http://www.copdess.org/enabling-fair-data-project

[3] https://rd-alliance.org/groups/software-source-code-identification-wg

6.4.2 Discussions

6.4.2.1 The challenge of rewarding people for different research outputs is not specific to software

Software citation and evaluation of software contributions are linked with wider issues in the evaluation of research outputs; the current focus on high-impact papers in well-known journals, e.g. Nature and Science, is well known. What is appreciated as research outputs will furthermore differ per research field, which have different ways of crediting and evaluating their research outputs.

More recent declarations, such as the San Francisco Declaration on Research Assessment (DORA[1]) and the Leiden Manifesto for research metrics (18), encourage movement away from metrics in the evaluation of research outputs. Following the guidelines of these declarations, research outputs such as software and data should be more important in research evaluation, although this should not lead to the development of another metric system to put a number on the work of individuals. This raises the question on how software should be valued, and how we should keep track of the (re)use of software. Software should be a first-class citizen[2] in research evaluation by funders, institutions and the broader research community.

In chemistry for example, a research group might publish a paper on the creation of a new molecule. Another research group uses that molecule and then adds a new chemical group to the molecule that makes it more stable. Should the original molecule creators be co-authors on the new paper, or should their paper be simply cited? There is no consensus on how to do this within the field - some seek to credit everyone; others seek to concentrate credit by limiting the number of authors. This is similar to software - at what point is the significance of the new contribution greater than that of the original contribution. When should people be added as contributors, rather than cited or just acknowledged?

Data is still undervalued, as are papers in low impact factor journals. Teamwork and collaboration are rarely rewarded in the current scholarly communication scheme. Whilst initiatives like CASRAI's CReDIT[3] have tried to define different types of contributor roles, they do not help us understand how to assign relative value to them. We should certainly seek to ensure that CReDIT accurately reflects the types of roles involved in the creation of software research outputs, but we still need to understand how to value them: is the person who wrote the code more important than the person who brought in the funding, or wrote the documentation?

Ultimately, we are dealing with the challenge of how we value the communication of research. Should we be valuing the ability, skills and experience of the person producing the software or valuing the software itself? Or both? This ties into questions of the unique value a contribution provides, and this is something that only others (e.g. peers) can identify.

[1] https://sfdora.org

[2] https://software.ac.uk/blog/2018-11-28-making-software-first-class-citizen-research

[3] https://casrai.org/credit

6.4.2.2 Including software in the "citation graph of research"

Ever since the Science Citation Index[1], and before then with manual indices, scientists have wanted to know which resources (originally just papers) cited which. There are a variety of reasons for this:

1. Looking at what an output relies on.
2. Looking at what an output has led to in order to do more science.
3. Looking at what an output has led to in order to do some evaluation of the output, and possibly the authors.
4. Producing summary statistics (generally in order to do some evaluation of the output, and hence of the authors). This point is frowned upon, e.g. by the Leiden declaration, but should not be confused with the previous uses.

This abstract graph of citations of science is now well-supported by both commercial (e.g. Scopus[2]) and free (e.g. Google Scholar[3]) tools. Other efforts of knowledge graphs including scientific data are: SciGraph[4] by Springer Nature, OpenAIRE[5] or Research Graph[6]. ORCID[7] information could be included as well and efforts around connecting different PID systems in FREYA[8].

This citation graph/linkage of research outputs can also be used by other stakeholders, such as those responsible for curation, impact management and career development. Note that use for evaluation (uses 3 and 4) tends to refer to the generic software, whereas uses 1 and 2 may well need to know the specific version. Hence the underpinning data must contain as much version detail as practicable, but there needs to be a collapsing mechanism (e.g. by referring to the name of a piece of software and meaning all of its versions) for uses 3 and 4.

The situation where software becomes a 'utility' and so underpinning that it is not mentioned, is seen as a natural progression for establishment of software. By way of analogy very few chemists state that they use water. The practice and situation in research software may not be as accepted as it is for chemists. The situation for software citation and when software becomes a 'utility' could well be different in different domains and institutions. To establish some consistency a concerted effort around the culture of software citation is required. This could I be linked to novelty vs impact, i.e. It should be noted that some types of successful software might go from ground-breaking (and hopefully cited) to utility (really impactful but not cited).

[1] https://en.wikipedia.org/wiki/Science_Citation_Index

[2] https://www.scopus.com

[3] https://scholar.google.com

[4] https://www.springernature.com/gp/researchers/scigraph

[5] https://www.openaire.eu

[6] https://researchgraph.org

[7] https://orcid.org

[8] https://www.project-freya.eu/en

6.4.2.3 How could we build software into the "citation graph of research"?

What would a citation graph of research that contained information about software look like? In order to build it, we need metadata for all research items. In terms of software, this means that citation metadata includes information of the direct dependencies, which in turn provides metadata about their direct dependencies and so on. Software dependencies in terms of citation includes reused packages and other research items such as papers and non-code contributions.

Software versions need to be included in the graph as nodes, as they may "cite" different sets of research. Only including concept nodes (i.e., root nodes for a version graph of the software) would impede resolution of links to research which used a specific version of the software.

Different views for this graph will be necessary for different use cases. Evaluating the research output of an institute may use a view of the graph that traverses it only down to software concepts rather than software version. Tracking re-use of software and its different versions would require deeper traversal of the graph.

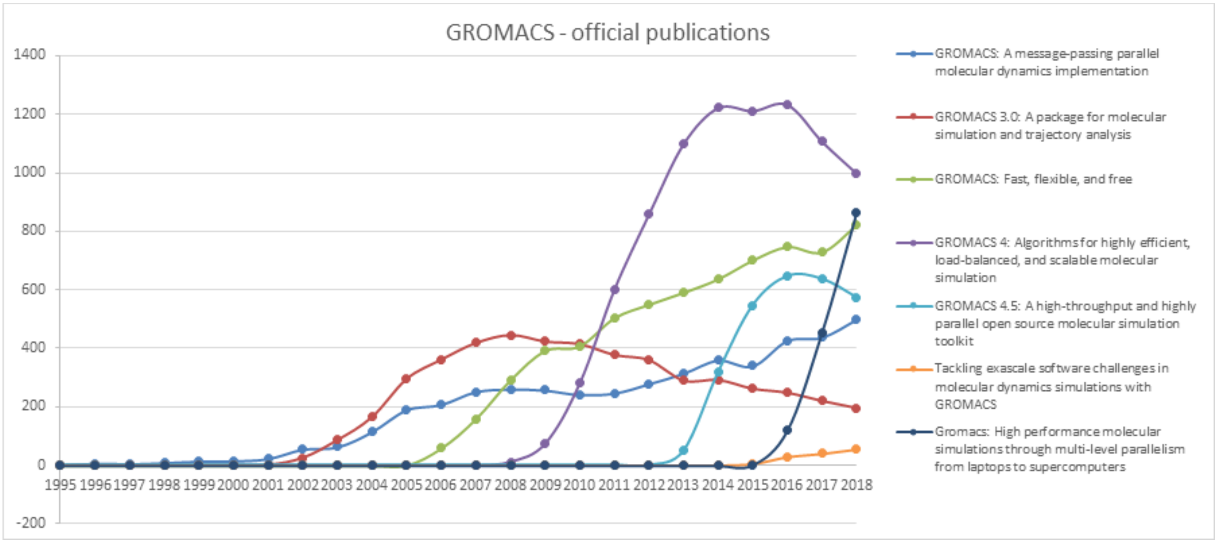

Figure1: (Google Scholar) Citations to various published versions of GROMACS software package. (source: Adriaan Klinkenberg)

It is important that any attempt to recognise contributions builds upon work already being done within the scientific community. One suggestion could be to work with projects such as CASRAI's CRediT or Metadata2020[1]. We need to work on defining what type of contributions we want to recognise and which of these should be considered core (e.g. a Dublin Core[2] for software) and when richer metadata terms are required (e.g. what would a contribution label look like for someone who identified a bug?). Existing standards and terms should be used where possible to facilitate interoperability with the current citation graph for research.

[1] http://www.metadata2020.org

[2] https://dublincore.org

6.5 Reproducible environment and containers

6.5.1 Introduction

Project Jupyter[1] is becoming one of the de facto ways to do, distribute and build reproducible research environments. It was recognised as the winner of the prestigious ACM Software Systems award[2] in 2017 (joining the ranks of Unix and Java in the process).

Jupyter and other systems have been combined to offer environments, deployment, scaling and services by using container-based approaches. Jupyter/docker-stacks[3] are Docker[4] containers for common user environments. JupyterHub[5] has Docker containers to ease deployment and provide isolated environments for users of a specific computing environment. Kubernetes[6] is a system which allows scaling JupyterHub from a single classroom to a large-scale massive open online course (MOOC[7]). BinderHub[8] and binder[9] are used to offer a service by turning a GitHub repository of Jupyter notebooks into a collection of live interactive notebooks.

It is important to understand the use case before selecting a container solution. It's important to converge on a solution for deployment and share this openly (e.g. Andrea Zonca's site[10]). There is always a temptation to build your own solution or software product to solve a particular container-based scenario, but this should be avoided when standard tools are available. Operational factors (such as using site reliability best practices), collaborative factors (such as documenting process), and human factors (such as performing blameless post-mortems) are essential for technically flexible and complex environments such as those involving containers.

Containers offer power and flexibility for distribution, deployment and scaling and they have additional challenges which should be taken into consideration when adopting them. There is an additional learning curve particular to containerisation and thus a software skills investment which also needs to be maintained. Additional skills in data access control, scaling and management of vendor lock-in also need to be taken into consideration.

Containers do open up possibilities such as allowing the deployment and use of scientific analysis. An example of this is the GenePatternNotebook[11] service. Interactive platforms that allow the chaining together of notebooks into workflows (e.g. nteract[12]) also make heavy use of containers.

---

[1] https://jupyter.org

[2] https://en.wikipedia.org/wiki/ACM_Software_System_Award

[3] https://github.com/jupyter/docker-stacks

[4] https://www.docker.com

[5] https://jupyter.org/hub

[6] https://kubernetes.io

[7] https://en.wikipedia.org/wiki/Massive_open_online_course

[8] https://binderhub.readthedocs.io/en/latest

[9] https://mybinder.org

[10] https://zonca.dev

[11] https://notebook.genepattern.org

[12] https://nteract.io

The increased use of notebooks (especially Jupyter) increases the use of containers as they make the deployment and sharing of notebooks (and thus research) far easier and they enable improved computational teaching methods[1].

6.5.2 Discussion

6.5.2.1 How containers are being used

Tools have been developed, such as INDIGO-DataCloud's project udocker[2] to allow the running of docker containers on High Performance Computing (HPC[3]) resources without the need for admin rights. The udocker system does not rely on Docker being installed but works with docker images/files. As an example, the udocker tool is being used by CERN[4] researchers to run old Fortran-based codes such as Mastercode[5]. By using the abstraction of a chroot[6] system it relies on a long-standing feature of Unix based systems which are likely to be sustained into the future.

6.5.2.2 Containers and sustainability

The ability to export a Docker image as a tarball[7] allows the system to be more independent of specific container technology. The tarball is a well understood tool that is not locked into a specific vendor format. The infrastructure and motivation for archival of such vendor free formats still need to be decided upon. Perhaps institutional (or public) repositories are the correct place, depending on the size of the tarball. Vendors (such as Docker) do facilitate the quick provisioning and deposition of containerised systems in the tarball format.

6.5.2.3 Containers and Data

Data should not be stored in a container format. The container format is volatile, there is no essential need to store the data in that format, and there are better formats suited to data archiving (depending on the domain or type of data). In many instances you want to move the computation (e.g. via a container) to where the data is.

A Binder service provisions a running computational environment from a repository (e.g., GitHub). You may use your own BinderHub instance to host an application where there are data access requirements that restrict you from running on a public service such as mybinder.org.

---

[1] https://jupyter4edu.github.io/jupyter-edu-book

[2] https://github.com/indigo-dc/udocker

[3] https://en.wikipedia.org/wiki/Supercomputer

[4] https://home.cern

[5] https://mastercode.web.cern.ch

[6] https://en.wikipedia.org/wiki/Chroot

[7] https://en.wikipedia.org/wiki/Tar_(computing)

6.5.2.4 Container limitations

Specialist hardware can need specialist support in systems that run containers, e.g. in the udocker tool it will support the host systems GPU[1] (e.g. for Tensorflow[2] based analysis), and ARM architecture. Currently, MPI[3] support for parallel computing is not complete (19). Executing MPI applications is possible but complicated, since the container has to have the same MPI libraries and versions as the host and the code has to be compiled inside the container, if this is not well managed then incompatibilities will arise and the batch system is likely to not function as expected.

6.5.2.5 Supporting Researchers

The Carpentries (carpentries.org) might be the right format to introduce researchers to containers. Although there are many examples on DockerHub, jargon and the learning curve present barriers to learners. The focus of a course would be good documentation practices, dealing with system constraints (e.g. where they don't have root access), APIs, automation (e.g. of publishing to DockerHub) and specific use case templates. Learning about containers is beneficial to researchers as some journals (e.g. SoftwareX) are starting to request containers of code to support peer review.

Domain specific efforts that bring together recommended software, technical infrastructure and a community do exist. An example of this is Pangeo[4] that are helping support and enable Big Data Geoscience research.

Another example is the DEEP-Hybrid-datacloud[5] which allows machine learning and deep learning modules to run on e-Infrastructure. This includes the support for GPU's. The system supports use cases around classification, monitoring and detection in a variety of domain areas such as health and remote sensing.

6.5.2.6 A note on containers versus virtual machines

Virtual Machines (VMs) simulate the entire computing environment from hardware to operating system. Containers only use the operating system of the host they run on, making them much smaller in terms of resources than a VM. A VM running a NumPy/SciPy based application would be slower than a container based equivalent due to the different resource requirements. Containers run as processes and use the services of the host whereas VMs run their own operating system services on top of virtual infrastructure.

VMs do allow more flexibility e.g. the ability to run a Linux based system on a Windows host. More guidance is needed around when to use VMs versus containers and how they can also interact[6], indications are that containers are replacing the use of virtual machines where containers are more suited to the task.

[1] https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units

[2] https://en.wikipedia.org/wiki/TensorFlow

[3] https://en.wikipedia.org/wiki/Message_Passing_Interface

[4] https://pangeo.io

[5] https://marketplace.deep-hybrid-datacloud.eu

[6] https://www.docker.com/blog/containers-replacing-virtual-machines

6.6 Archiving

6.6.1 Introduction

The Software Heritage Archive (SHA[1]) aims to collect, preserve, and share all software that is publicly available in source code form. Using this foundation applications can be built for cultural heritage, industry and research.

Software is an essential part of our lives. Given that any software component may turn out to be essential in the future, SHA collects all software that is publicly available in source code form. Software of particular interest should have specific support for curation activities, and this is under the preservation remit of the archive. The full development history of the software products that are archived are also stored allowing unanticipated future use.

Preserving software, preserves our technical and scientific knowledge. The archive itself will openly describe its technical architecture and processes and be free and open. This important work requires nurturing a network of peers and mirrors to share the responsibility of maintaining availability of the software that is collected.

The archive is building the largest archive of software source code ever assembled. It will index, organize, make referenceable and accessible all of this precious heritage. It will provide unique identifiers to the software components. This will ensure that a resilient web of knowledge can be built on top of the SHA.

Software Heritage will foster the emergence of a variety of services, ranging from documentation to classification, from search to distribution, to release all the potential of this Library of Alexandria of Software.

The SHA currently contains over 100 Million projects, comprising over 7 billion files and 1.5 billion commit histories.

6.6.2 Discussion

Archiving software is intimately related to archiving data. For common data formats (e.g., pdf, jpg) different software programs can be used to access the data. However, for domain specific data, specialized data formats may only be accessible with specialized software or data may be integrated into the software; in such cases the reusability of software is essential for the reusability of the data. Data may also be sensitive to which version of a piece of software can handle it; especially for proprietary formats and software.

There is an acknowledgment that what we can do with a software archive will change over time. Similar to preserving old books where the language changes: there is intrinsic value to it, but what you can do with it will change over time.

[1] https://www.softwareheritage.org

There are various reasons for archiving software:

- Research software: to be able to recreate research results or modify the software to generate new research results.
- Reproducibility of software: being able to recreate and experience old software; sometimes all we have are the executables.
- Heritage perspective: study of the code itself (software archaeology, software phylogeny) as a new field of research and science.

It is important to consider the goals and designated communities in the preservation choices that are being made. These goals will translate into different levels of preservation:

1. Archiving the files associated with a software project.
2. Archiving all the preservation metadata needed to reconstruct environments (without necessarily keeping all the components).
3. A platform that allows continuous execution in the future.

For each of these levels there are additional resource requirements. Also, the level of preservation can change over time depending on the goal with which you archive software.

These three levels can also require different places to put or archive software:

- Archives (e.g. SHA).
- Development and distribution platforms (e.g. GitHub).
- Domain & community software deposits.

The goals will also translate into different approaches with regards to selection and appraisal. We can differentiate between archiving based on selection and curation and (automatic) archiving of complete collections (e.g., "all of GitHub"). In the latter case quality checking would remain on anyone using this type of archive. Both kinds of archiving can be valuable for different purposes.

For archiving based on selection and curation, minimum requirements for ingest must be formulated. For example, you may include criteria such as: the documentation to be of a certain quality, tests being available and at the point of ingest that the software compiles and runs.

6.6.2.1 Challenges

Preserving software and data is a process rather than a final destination. It is also a collective effort where transformation for the safe of preservation cannot harm perceptions of authenticity.

Best practices in software can also be thought of as 'pre-archiving' skills, they make the process of preservation more effective.

Developing tools to aid preservation is an ongoing need; e.g. web archiving tools are needed to address what's missing from an archive.

There are deep challenges related to archiving and preserving knowledge that apply equally to software; how do you preserve the related implicit, social, contextual and embodied knowledge to allow software and data reuse in a way that is mindful of the original environment in which it was created.

Modern technological advancement can become preservation threats and need to be actively managed if used; the risks in commercial cloud infrastructure to preserve world heritage could become a concern (e.g. the Software Heritage Archive currently runs on Microsoft Azure[1].

6.7 Resourcing and planning

6.7.1 Introduction

To examine how organizations resource and plan work, we start with a case study. The Edinburgh Parallel Computing Centre (EPCC[2]) is the novel computing centre at the University of Edinburgh. It is a self-sustained unit that has been around for over 25 years. It has over 100 members of staff and around £13m turnover per year. It has a multidisciplinary focus and is funded from a large number of sources from the UKRI[3], to EU projects, industry and others. It is involved in a large spectrum of activity and has a critical mass of expertise. It is involved in technology transfer, European projects, access to facilities, training and HPC research.

With such a large organisation focused on software activity the pressures are similar to other organisations:

1) Money is needed to maintain and support activity.
2) People are needed to keep things running; recruitment, retention and satisfaction are key aspects that need management.

The EPCC takes a growth-oriented perspective to staff satisfaction. By giving staff exciting projects that nurture their talents and reward them appropriately, staff satisfaction is actively managed.

The EPCC capabilities are developed through work on collaborative projects, developing solutions and people and hiring to fill in gaps in capability.

Challenges remain around resourcing projects, as a large centre it has more flexibility and people to call on, sometimes it has to balance conflicting requirements. There is also the EPCC planning system which optimises resource allocation based on spend.

Thus, through recruitment and retention of staff and a focus on their personal development and frequent planning to optimise projects priorities, resourcing and planning, successful delivery is realised.

---

[1] https://azure.microsoft.com/en-gb/overview/what-is-azure

[2] http://www.epcc.ed.ac.uk

[3] https://www.ukri.org

6.7.2 Discussion

In all organisations that develop software, a sustainable workforce promotes the sustainability of software; the knowledge developed in projects is lost when the workforce moves away. This is sadly often the case in academia as many projects struggle with retaining staff in the long term and therefore knowledge is lost. Some reasons for this include policies which prevent hiring staff on a permanent basis and employment being dependent on project-based funding.

Software can be sustained when the software developed in one project can continue to be used in subsequent projects. In order to promote the sustainability of the workforce who develops research software, software must be recognised as being an important part of the research process.

Recognition in the current academic system is largely based on publication of academic articles. There is little recognition of the value of developing high quality research software. Possible mechanisms to achieve recognition of software are case studies, software reviews, peer-review of code, software papers, software reuse, software citation and quality ratings for research software.

Some software can be sustained from grant to grant, but institutional planning and the intentional use of resources is more likely to lead to sustainable software over the long term. Organisations need to make a conscious choice about which software to sustain: some software will be sustained because it keeps being part of a grant; other software is sustained because of a decision to keep it alive and may be funded, for instance, via institutional funding.

7 Legacy & Recovery

7.1 Re-use Metrics

7.1.1 Introduction

When we mention (re)use metrics for software, what do we mean? Is it the specific number of times something is (re)used? Why are we choosing to add value to this event - in fact what should we count as (re)use? If (re)use is our metric, then it's a number that represents the accumulation of certain events. We come back to the problem of where the number comes from. The (re)use metric needs to be presented in a clearly understandable way if it is to be useful and meaningful. We use the term (re)use to cover use and reuse as the terms are so closely aligned.

With a clear and commonly understandable definition of (re)use, it will be possible to think about impact metrics for software (e.g. like a software driven h-index). (Re)use metrics should indicate that the purpose for citation is known. FAIR's application to software is relevant to (re)use (see 6.2 on FAIR Principles for Software) clearly as the R stands for reusable.

The current state of the art in software metrics is achieved via services such as Crossref EventData[1], the Data Citation Index[2] and ScholeXplorer[3] that focus on tracking software related DOI's in the reference section of publications. But due to the idiosyncratic nature of software citation-overlaps are hard to find programmatically and require manual methods (e.g. using a combination of Google Scholar and other services) - which of course are not scalable and lead to references being missed. In short, today's approach requires you to mimic the famous British investigator, Sherlock Holmes[4] to find all the (re)use cases. It is hoped new approaches that employ more pattern matching approaches (e.g. Machine Learning) may yield new solutions and better results in the future.

Displaying software metrics well is key to them being trusted and understood. Summary statistics for the citation and usage of software as a headline figure are important. However, more detail should be available for those who want to explore what the figure means. As well as user driven linking between publications and software, curated citation metrics are needed.

Zenodo now has integration of data from Altimetric about the impact of an article via citation and what exactly that impact looks like (e.g. number of tweets, number of news outlets this was picked up by) and views, downloads and details of the performance of each version. Clearly this is for generic digital objects resolvable by a DOI, there might need to be specific changes for software that is referenced by a DOI e.g. such as where software is mentioned in publications (as a proxy of the part it played in research).

[1] https://www.crossref.org/services/event-data

[2] https://clarivate.com/webofsciencegroup/solutions/webofscience-data-citation-index

[3] https://scholexplorer.openaire.eu

[4] https://en.wikipedia.org/wiki/Sherlock_Holmes

7.1.2 Discussion

7.1.2.1 (Re)use metrics and dimensions

There is a debate around the difference between software use and reuse. For the purposes of this section we will treat them synonymously and hence refer to software (re)use (see chapter 3 of (20) for more discussion on the term reuse).

(Re)use metrics will differ depending on the audience of that data; the original developers, funders, policy makers and people who are investigating which software to choose will all have different needs. The original developers will focus on how their software is being used by others. Funders will want to track the impact of their funded work. Policy makers will want to understand the wider impact of software in the research space. People investigating which software to use will look at whether the software is maintained and the size of the user base.

Thus, the importance of different measures depends on the stakeholder in question. Projects that can be used to aid determining quality are Libraries.io SourceRank (e.g. an analysis for Python's matplotlib package[1]) and Recoda[2]. Adherence to Minimum (21) standards are also useful when evaluating software.

The validity of metrics for various stakeholders will vary and there is a need to educate people on interpreting the metrics that do exist. Any training in this area should include when metrics can and should be used, and when they might provide incorrect information for their particular needs. An example of a potentially incorrect comparison is when comparing software at different maturities, that have existed for different lengths of time.

There are other endeavours trying to define metrics. The CHAOSS group is defining metrics[3] more generally for open source software, which includes wider themes such as risk, value and diversity alongside the traditional aspects around code development. These matters are likely to be relevant for different stakeholder groups in the context of (re)use.

7.1.2.2 Metrics dimensions

A number of potentially useful metrics along different dimensions include:

1. (Re)use metrics
2. Quality metrics → beyond scope
3. Reproducibility metrics → beyond scope

All metric dimensions overlap with each other to a certain extent: some reproducibility metrics can be used to measure software quality, some (re)use metric may be used to measure software quality.

[1] https://libraries.io/pypi/matplotlib/sourcerank

[2] https://github.com/hansendx/recoda

[3] https://chaoss.community/metrics

In the following sections, we concentrate on discussing (re)use metrics.

7.1.2.3 Metrics dimensions for (re)use

We highlight different categories of metrics for (re)use.

7.1.2.3.1 Project Details

Current (re)use metrics for software include:

- Downloads, licenses sold, queries (services), users (e.g., if interface available via web site)
- Forks (although this relies on services being sustained e.g. GitHub, where it was claimed by one of the participants that currently 50% of research software activity takes place)
- Contributors: in/out the project
- Latest commits
- Versions
- Status (maintained, deprecated)
- License (to cover whether it can be reused)
- Degree of sustainability

7.1.2.3.2 Users and developers

There are also proxy metrics which would allow us to infer the likelihood of (re)use by the current ease of (re)use. There is a difference here between (re)use by users (who make use of the software as is) and by developers (who may modify the code or include the code in their own projects). But it should be noted that both users and developers are valid to consider with respect to the software's (re)use in research as an aggregate group.

This would include measures such as the availability or use of a:

- Readme
- User documentation
- Unit testing
- Use of continuous integration
- Metadata and links to related resources (e.g., datasets with which the software has been used on, other software that has been built upon the software)
- Citation metadata

7.1.2.3.3 Future metrics

In a future (ideal) situation where a proper citation for software exists (i.e. where software is uniquely identified and has machine actionable metadata), the following could be found out from citations about the software:

- How much it's (re)used (quantity)
- Who is (re)using it
- What it's (re)used for
- When it has been (re)used
- Whether this use has "impact" (though this is hard to objectively define)
- Where it was developed and by which institution(s)

7.2 Legacy

7.2.1 Introduction

There are many issues that have had to be faced in the case of "legacy" software. The experience gained covers the mainframe era, dominated by proprietary operating systems, the age of minis and micros, clusters and farms and finally moving to grid and cloud computing. Modern tools, technologies and methodologies may mean that some of the issues seen during the past decades will be less of a problem in the future, but it would be naive to imagine that the "steady state" will prevail. Some of the key issues that have been faced in the past and continue to be faced by current projects are highlighted below.

Operating systems change. Programming languages change, what are legal code constructs today may not be in the future. Code repositories and build systems also change. How do we prepare for these inevitable changes and help ensure that today's code is preserved for the future along with any necessary configuration or other data? Is this even possible if people from the original project are no longer available? If software is essential to understand and use particular data, does this mean in some cases, we can only preserve data for the length of a person's career?

To preserve software for future re-use, we need more than just the source code. Can we agree on a checklist of the things that are required and/or desirable? This should include documentation of the software and associated methods, build instructions and validation suites and configuration data. How do we rectify the situation where these guidelines were not followed in the past?

Private code can be lost, e.g. when a user leaves their storage maybe wiped. We need to encourage code associated with analysis (e.g. in a PhD or for a publication) to be captured. A way to motivate this would be to make it a requirement that supporting code be made available alongside the analysis (e.g. make it a requirement while undertaking a PhD or submitting a publication).

Further analysis of the problems faced can be found in the report on software preservation and legacy issues at LEP (22).

7.2.2 Discussion

7.2.2.2 Legacy

Digital technology is constantly changing; operating systems change, programming languages evolve, build systems change. This creates a challenge for reusing software in the future. What data should be preserved to ensure the software will run? How do we preserve expertise to understand and run legacy software? Are there lessons to learn from the nuclear waste management community who make no assumptions about the future?

The process of preservation requires regular evaluation and active technology watch to understand what is happening around us in terms of supported file-formats and software that's backwards compatible. The Open Archival Information System (OAIS[1]) reference model that guides the long-term maintenance of digital repositories includes technology watch. Those responsible for preserving software need to know about the large changes that have happened in software technology in the past (e.g. word length) so that they are aware of what might happen in the future.

Preservation activities are not without their pressures and oversights. The Open Preservation Foundation (OPF[2]) website for example lost many expert comments during its migration. Pressures do exist between project-based approaches that focus on time, quality and scope that might prioritise new features over authentic preservation.

Making data and software more usable at the time of deposition is time consuming and there should be incentives to promote the quantity and quality of this type of publication.

'Slow Science' might be a better approach to defining things as complete, to allow time to document, publish data, software and working processes in a way that others can reuse and reproduce results rather than being given the green light to move to a new project.

Software does present additional complications over data. New features are added, bugs are fixed, and it needs to be adapted to new environments. For software to be sustained it needs to be clear who is responsible for these activities.

There are technological methods and artefacts that aid future preservation and should be employed in active projects. Virtual machines, containers and emulation can aid in deploying and running the software by preserving its context. In addition, test cases, quality checks, clean code, reliable documentation, any handover documentation, produced by the original developers (if they have moved on) will aid the verifiability and authenticity of the code and any preservation actions.

These methods will aid in research knowledge being preserved and not lost every time a student or member of staff leaves a group. Encouraging or mandating these best practices is foundational to

[1] https://en.wikipedia.org/wiki/Open_Archival_Information_System

[2] https://openpreservation.org

Open Research[1] and trustworthy research[2]. Incentives and recognition for practicing such techniques should be put in place.

7.3 Cultural Heritage

7.3.1 Introduction

7.3.1.1 Software as heritage

At present very few cultural heritage institutions (CHIs) in the Netherlands consider software as an integral part of their collections. Digital collections in the Netherlands are by and large comprised of standard archival formats that can be migrated to other file formats in case this is needed in the future. Software is hardly mentioned in policy documents, whether collection policy or multi-annual plans.

However, research and experimentation are taking place at various institutions that are increasingly concerned with software as heritage. By and large, two approaches to software as heritage can be discerned. First, where software itself is the archival object because it represents socio-cultural value. An example of this are computer games, or software that was published bundled with a paper publication. A second approach is where software is conditional for sustainably providing access to specific file-types. Consider for instance CAD and QuarkExpress[3] files at an institute for architecture and design.

There is a growing awareness that in order to render specific files, but also software executables authentically, there is a need for access to the software-environments that enable it. Within the research projects mentioned the focus is on emulation as a strategy for preservation and access. There are interesting developments around Emulation as a Service and technology has been developed that will lower the threshold to emulation for a larger audience. Within the Dutch Digital Heritage Network (NDE[4]) a project on software preservation runs until the end of 2020.

7.3.1.2 Collaboration around the development of tools within the cultural heritage domain

The OPF was founded in 2010 (initially as the Open Planets Foundation) to sustain the results of the EU-funded Preservation and Long-term Access Through Networked Services (PLANETS[5]) project. Its mission is to 'enable shared solutions for effective and efficient digital preservation'. OPF's main activities comprise co-ordination, management and technical support of open-source software development, training activities and knowledge sharing. OPF is primarily funded by its members (26 as of February 2020), with additional funding from projects and consultancy.

[1] https://en.wikipedia.org/wiki/Open_research

[2] https://en.wikipedia.org/wiki/Scientific_integrity

[3] https://en.wikipedia.org/wiki/QuarkXPress

[4] https://www.netwerkdigitaalerfgoed.nl/en

[5] https://www.planets-project.eu

The main software tools supported by OPF[1] are JHOVE (file format validation and feature extraction), Fido (file format identification), VeraPDF (conformance checking for the PDF/A standards) and Jpylyzer (JPEG 2000 validation and feature extraction).

The Jpylyzer tool is a good illustration of OPF's role in making project results sustainable. Jpylyzer was originally developed by the National Library of the Netherlands (KB[2]) as part of the EU-funded SCAPE[3] project. After the end of the project in 2014 it was adopted by OPF. Even though Jpylyzer's core development still takes place at the KB, OPF has been largely responsible for setting up continuous integration workflows, automated static code analysis for pull requests, unit testing, and packaging for various platforms.

Thanks to OPF, Jpylyzer has reached a level of maturity that wouldn't have been possible if the KB had developed the software alone. In addition, OPF's work on continuous integration and automated testing has made the release process considerably simpler. Also, the hosting of the Jpylyzer website under the OPF domain has been important for the visibility of the tool. Last but not least, the risk of the Jpylyzer code base becoming orphaned has been greatly reduced, since thanks to OPF it does not depend on a single developer or institution.

7.3.2 Discussion

The preservation of cultural heritage items, such as old games, media art or live electronic music, requires working with producers and developers. The type of metadata, data and software one needs to preserve are important for cultural items. But one needs to know how much to store - how much is enough, it’s clear that digital fantasies of perfect reproduction, endless resources, perfect scaling of activities, cheap and kept forever are just that - fantasies.

Information is lost - how do we plan with this in mind. Permalink patterns change, funding decisions can mean certain websites are lost. Can we afford to lose some convenience for the sake of preservation to maintain e.g. websites in a sleeping rather than active states?

Training is needed to make things with preservation in mind so we can avoid preservation debt when developing software in research and for the cultural sector. Credit and measurement are needed to help incentivise making things more preservation ready.

The emulation approach does require a measure of how authentically the software is rendered; this itself is an artefact that needs to be preserved and sit along the software.

7.3.2.1 Emulation

Collaboration across institutions and domains that involve emulation require conversations about copyright, shared infrastructure, shared vocabularies and decentralised repositories.

[1] https://openpreservation.org/products

[2] https://www.kb.nl/en

[3] https://scape-project.eu

There is a need for institutions to work together to share resources needed for emulation. One possibility is to have a national/regional institute responsible for providing virtual machines. This would require significant provisioning, hardware and support and it would serve individual researchers, institutions and software owners with a valuable service.

There are copyright initiatives by UNESCO which attempt to balance interests of authors and the interest of the general public. An example of issues with copyright is the old versions of Microsoft Windows where the company might not even own all of the copyrights to the software they published which makes it complicated to use old versions. In 2018 Microsoft released MS-DOS[1] v1.25 and v2.0 as Open Source under a license similar to MIT. Also a re-implementations of Windows may be able to compensate for software requiring older versions of Windows (e.g.ReactOS[2]).

[1] https://github.com/Microsoft/MS-DOS

[2] https://reactos.org

8 Policy & Quality

8.1 Quality Metrics

8.1.1 Introduction

Quality measures for Research Software are a relatively new concept in software dependent research projects. With the advent of the European Open Science Cloud (EOSC) and its requirements (23) for Technology Readiness to ensure the services offered are of sufficiently high quality for the users, i.e. individual researchers and communities, this is changing, albeit slowly. Standards can appear abstract, the question then becomes what to actually measure in practice and how this aligns with the real practice of software development. Agreement on specific metrics and availability of tooling are part of what will make any common approach feasible and implementable.

8.1.2 Discussion

A wide variety of software quality measures already exists, for example the software product quality model definitions of ISO 25010[1]. From there, suitable metrics or KPIs can be defined and used in particular cases.

Research software does not typically adhere to such quality measures. However, the EOSC requires services listed in its catalogue to be of sufficient technical maturity; this requirement makes it necessary to provide appropriate evidence of maturity. One way to provide such evidence is by adhering to software quality measures geared towards research software, such as the CESSDA Software Maturity Levels (24), quality assessment (25) and the Netherlands eScience Center guide[2].

For scientific software at least, there is always the need to find a balance between what is needed and useful and what is actually attainable. Researchers are not necessarily trained in software engineering and are often time constrained for improving the quality of their research software. Ideally, research software activities should be considered as another component of conducting research (see section 6.4 on citation and reward systems).

Measuring research software quality can serve a number of purposes: being able to provide evidence for quality can encourage trust in the tool, its functionality and continued availability. Over time, it can serve as justification for time spent on quality improvement, in particular when changes are transparent for users and managers. Metrics aid the planning and prioritisation of technical work which helps reduce costs by addressing technical debt early on.

Depending on the purpose of research software quality metrics, a number of possible quality metrics are worth considering. Among the simplest indicators are:

- Use of version control.

[1] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[2] https://guide.esciencecenter.nl

- Presence of a Readme file, e.g. containing installation instructions.
- Quantity and understandability of documentation.
- Consistent coding style; verified by linting.
- Performance measurements.
- Intellectual property information (e.g. copyright notices).
- Availability of archived copies with persistent identifiers.

Provenance and plagiarism checks can also be relevant information.

8.2 A European Software Sustainability Infrastructure

8.2.1 Introduction

Two before the WOSSS19 workshop, a proposal for a design study to look at the conceptual and technical design for a European Software Sustainability Infrastructure (EuSSI) was submitted to the European Commission. While this was well-reviewed, it was not funded as part of a competitive call. In the intervening period, several things have moved on, including the rise of the Research Software Engineer[1] and a number of software sustainability initiatives starting up worldwide. However, all these initiatives have different strengths and gaps. This means there is still an opportunity and space for a transnational infrastructure to support software sustainability. Key questions that need discussion concern what EuSSI could do better than individual software sustainability initiatives, who should be involved in the creation and delivery, and what's the right balance between the different types of infrastructure: people, training, tools and services.

8.2.2 Discussion

8.2.2.1 Why a European Software Sustainability Infrastructure (EuSSI)

Having a European infrastructure is needed for:

- Knowledge transfer between different domains and initiatives.
- Promote partnerships and collaboration between initiatives and the research software community at an EU level.
- Enabling cross-European projects to work on the sustainability of already-created software.

For the EU to consolidate its position as a global leader in this field, there needs to be a coordinated approach. We can already see that not all EU member states are at a similar maturity level. An EuSSI can therefore coordinate the onboarding of member states who lack skills and capabilities in this area.

[1] https://society-rse.org/about

An EuSSI can ensure return on investment for infrastructure developed in H2020 projects[1], making sure that knowledge developed is not lost, and that software created remains reusable and accessible after projects finish. It will lay the foundations for a sustainable European body to coordinate activities, exchange knowledge and teach on scientific software developments in a collaborative environment.

Regarding education, the EuSSI can also play a role in identifying the core competencies needed by researchers/developers and create the educational routes required for upskilling. It is important that not only software developers and researchers within ICT-focused disciplines have knowledge of software sustainability, but also broader scientific disciplines who are currently less aware of this topic (in order to promote best practices from the start).

An EuSSI could support European researchers and promote and align best practices with and among European software-related enterprises. Together with SMEs it can act as an incubator for new developments in the field of science and for the private sector.

However, coordination at a European level could be confusing and extremely challenging, as there are already initiatives here and there. A mechanism enabling networking between existing initiatives would be useful and thus a pre-study of what the achievable goals would be towards establishing an effective EuSSI.

8.2.2.2 What will an EuSSI provide

The EuSSI will be structured on existing pan-European infrastructures such as ELIXIR[2]. While these infrastructures have different operational models, EuSSI will have a regional approach. The current SSI services[3], such as the Fellowship Programme, RSE Policy efforts, outreach, Research Software Support/Consultancy, will serve as a starting point, but will be run at a regional level, and coordinated at the European level, to share best practices, templates and procedures as well as explore strategic development of services. Given its platform there will be a new activity where we engage EuSSI with other larger European initiatives in Open Science[4] such as EOSC, other European Strategy Forum on Research Infrastructures (ESFRIs[5]) and European Research Infrastructure Consortiums (ERICs[6]); building and tailoring services to regions and infrastructures.

EuSSI will provide a better and unified voice for working with the European Commission than purely national efforts. It will be a single point of contact for projects or people both in Europe and outside who want to work with European partners in the area of software sustainability, or who need to comply with European policies. EuSSI will provide coordination for other European-wide projects for software sustainability issues; the EURISE Network (eurise-network.github.io) has already stated an interest in cooperation with an EuSSI.

[1] https://ec.europa.eu/programmes/horizon2020/en

[2] https://elixir-europe.org/about-us/how-funded

[3] https://www.software.ac.uk/programmes-and-events

[4] https://ec.europa.eu/research/openscience/index.cfm

[5] https://www.esfri.eu

[6] https://ec.europa.eu/info/research-and-innovation/strategy/european-research-infrastructures/eric_en

EuSSI will provide a neutral territory to help share best practices between different efforts and an organisational approach based on collaboration, sharing and trust between partners that has worked well for the current version of SSI in the UK and the project and programme management expertise which has been commented on positively by the community that SSI currently interact with. The SSI is commended by its community members for using the latest thinking and practices and innovating in terms of collaborative working, efficient use of time and giving a voice to community members.

EuSSI will take an approach to authority in this area which is more geared towards de facto acceptance rather than de jure enforcement. Allowing the community to be comfortable to take things onboard and then adapt to their needs as well as being receptive to their ideas and incorporating this when updating best practices. The sense of renewal of practices and approaches which characterise SSI will be taken on board for EuSSI - adapting to the changing needs of the communities that are served while taking a collaborative approach.

Research domains have varied software development needs. Providing canonical best practices would be a bit heavy handed or authoritarian and would likely be received with resistance from individuals. However, offering baseline "sensible" defaults for starting points for development would have value for small projects and individual researchers. For example, a doctoral student looking to begin a software project could use the baseline defaults to get started - to begin the project with sustainable software practices.

These sensible defaults could take various forms, including but not limited to guides to existing European computing resources and Research Infrastructures, checklists of open source tools that are well maintained and widely used, cookie cutter repositories with default configuration files for testing, documentation, and code quality tools. Similar to the Carpentries work, a workshop teaching the application of these resources and sensible defaults for early career researchers would provide the researcher with valuable skills which would free up the researcher's time to focus on research pursuits over software engineering details.

In many cases, it is not enough to provide the best practices and tools to researchers but also the infrastructure and services to teach or use them.

8.2.2.3 What would be in a case for supporting an EuSSI

Operational experience from the UK-based SSI has shown that there is a definite need for an infrastructure that has a neutral, cross-cutting, interdisciplinary approach to software sustainability. One that provides evaluation, consultancy and proactive advocacy in the field.

EuSSI builds on this experience to facilitate Europe-wide knowledge exchange and collaboration of project-based as well as institutional, regional, national and supranational initiatives. It will bring stakeholders together to establish common baseline guidance and enable practices of sustainable software development (e.g. FAIR and open software), management and maintenance. As a facilitator, aggregator and distributor of strategies, policies and practices it will drive wider adoption of good practices for software sustainability across regions and disciplines, scaling and adapting the work of the UK SSI to the European stage.

Any case for an EuSSI should cover the following topics to highlight the value proposition:

- Increased knowledge sharing and benefits realisation for the European research community; more reuse and less re-invention.
- A unified and neutral voice for working with the European Commission than purely national or domain focused efforts.
- Interface and co-ordination with the existing funding landscape and initiatives as software enabled research is ubiquitous.
- Development and promotion of baseline practice and standards with training to aid researchers and large projects across domains.
- Provide European tailored advice in terms of General Data Protection Regulation (GDPR) compliant services (e.g., Cloud) and software related ethical and legal concerns.
- Promote the use of applicable European research infrastructures and services.
- Guidance on the best state for software before, during and after the project via guidance on software aspects in Output Management Plans and in line with FAIR principles and Open Research methods.
- Scale proven initiatives such as the Fellowship Programme (26) across Europe.
- Build a network of domain researchers with a software focus and research software engineers to help advocate and participate in promoting and formulating best practice.

9 Final panel discussion

Panel participants: Neil Chue Hong, Carol Willing, Catherine Jones, Daniel S. Katz and Patrick Aerts
Chair: Shoaib Sufi

To close the workshop, a panel discussion was organised giving experts in software sustainability a platform to share their views. The topic of the panel was "Areas of future work in software development, legacy & policy". The panel chair prepared a set of starting questions, which were answered by panellists and followed up by questions from the audience.

9.1 What existing workshops/conferences/efforts are taking place in the space or linked to software sustainability and what is/are their focus

Neil: Small initiatives on different domains (such as humanities) are starting to talk about software sustainability, but they are still disconnected.

Carol: Open source communities are the ones talking mostly about sustainability for some time.

Catherine: <*reporter missed her answer*>

Dan: There are three events that come to mind:

- NumFOCUS[1]: they have an event which brings together projects sponsored by NumFOCUS to share their experience on sustainability.
- WSSSPE[2]: projects which were more successful were the ones which have a community behind them and WSSSPE gives them a platform to talk about sustainability.
- Super Computing (SC[3]): where the panel on sustainability was similar to WSSSPE.

Patrick: We have WOSSS; the eScience symposium -- all the topic the eScience Center is involved in are discussed there, Plan-E[4] and E-plan also always discuss FAIR, EOSC, etc -- Science Europe also discusses this at a policy level; finally (in the Netherlands) NDE (Netwerk Digitaal Erfgoed) where the national library and other cultural institutions discuss software and sustainability.

9.1.1 Follow Up: How about the HPC field? How much do they care about sustainability?

Dan: On a negative note, HPC centres at a high level seem not to care about it very much (beyond getting funding), though the individual developers at these centres certainly do.

[1] https://numfocus.org

[2] http://wssspe.researchcomputing.org.uk

[3] http://supercomputing.org

[4] https://plan-europe.eu

On a positive note the United States Department of Energy has supported an initiative on sustainability (IDEAS[1] and BSSw[2]) for a few years.

Carol: HPC world does have code which is 50 years old - it is not 50 years old, but it has been sustained for 50 years.

Patrick: The HPC field is very advanced in many ways, but not necessarily on sustaining something -- they focus a lot on making things run as fast as possible and do not care about anything else.

9.1.2 Follow Up: What is the overlap between WOSSS and the International Conference on Software Maintenance and Evolution (ICSME)?

Patrick: It seems the education and industry are very disconnected; industry spends a lot of effort in re-training people coming out of universities.

Neil: There is a need for more interaction between industry and research.

Carol: Languages themselves have different time spans. What works in research and what works in practice has a different focus. It is a balance and it is important to keep up to date in what happens in both fields.

Dan: Industry and research work differently. When we have brought people from industry to research, it has not been as successful as we would like it to. What works in industry does not necessarily work in research.

9.1.3 Follow Up: Are computer science undergraduates learning best practices in their courses compared to what they need in practice?

Dan: They usually have good bases and are usually able to pick up skills quickly.

Patrick: It changes between universities, and between countries. It is mostly a mind change - undergraduate students should learn that data (and software) are valuable and they should learn that at an early stage.

Catherine: We should not only focus on computer science skills, but also on developing essential skills across other domains as well.

Carol: Look for students who have completed large open source projects.

[1] https://ideas-productivity.org

[2] https://bssw.io

9.2 How have you experienced this workshop? Where does it fit in the landscape?

Neil: Good coverage of different sectors - computer specialists, libraries, archives, etc. It brings together people from different domains. How often should this be? That is a question for the audience.

Carol: Cross pollination across communities is important.

Catherine: I also like the diversity. Frequency, every 2 years seems right.

Dan: Positive -- nice diversity (WSSSPE does not have such diversity for instance); less positive -- the name is not necessarily meaningful. Another question is what impact does producing a report have?

Patrick: Probably biased about the name, because I like alliteration, so I like the 3S's in WOSSS. Historically, the first one was organised in Berlin in 2015; in 2017 DANS organised WOSSS together with SSI, this time also including the Netherlands eScience Center. I would ask the audience to give feedback.

9.3 How do we engage with domains that do software sustainability well and domains that could improve?

Dan: I don't feel domains do it inherently better than others, but rather people in one domain or another who are more skilled than others.

Catherine: Do not underestimate domain differences. We need to go to different domains and work with them.

Carol: Some people are passionate about it, and these are the ones who are easier to engage.

Neil: Some domains do better because of better organisation. Better organisation helps find people who are enthusiastic and who we would like to engage.

Dan: Recommend book: Crossing the chasm[1].

Patrick: Very impressed by skills on domains: KB, Beeld en Geluid[2], etc. Really interesting learning from different communities.

[1] https://en.wikipedia.org/wiki/Crossing_the_Chasm

[2] https://www.beeldengeluid.nl

9.4 What changes would you like to see in the community and in this workshop in 2 years?

Dan: I would like to see more people in the communities to know what the challenges are in the different communities.

Catherine: More diversity in terms of ethnicity.

Carol: Explanation of background in badges.

Neil: Better understanding of what everyone is doing.

10 Agenda: Workshop on Sustainable Software Sustainability 2019 (WOSSS19)

10.1 Day 1 - 24 April 2019

12:00-13:00 Arrival & Lunch (Sponsored by SSI)
13:00-13:10 Welcome by the Local host (Peter Doorn, DANS)
13:10-13:20 Welcome by co-organiser (Neil Chue Hong, SSI)
13:20-13:30 Welcome by co-organiser (Carlos Martinez-Ortiz, NLeSC)
13:30-13:40 Goals and format of the workshop (Patrick Aerts)
13:40-13:50 Break-out session logistics explained (Shoaib Sufi)
13:50-15:00 Introductions to the "Preventing new legacy - part A" topics - 1

- Software development (Robert Haines)
- FAIR Principles for software (Anna-Lena Lamprecht)
- EOSC, FAIR & Software (Mario David)
- Software deposition routes (Cees Hof (presenter) & Carlos Martinez Ortiz)
- Software Directories (Jurriaan Spaaks)

15:00-15:20 Break
15:20-16:50 Break-out discussion & writing sessions - 1
16:50-17:25 Reporting back from breakout sessions - 1
17:25-17:30 Wrap up and close Day 1
19:00-21:00 Workshop Dinner (sponsored by DANS)

10.2 Day 2 - 25 April 2019

09:00-09:05 Welcome to Day 2
09:05-10:00 Introductions to the "Preventing new legacy - part B" topics - 2

- Software Heritage Archive (Roberto Di Cosmo)
- Citation and rewards systems (Daniel S. Katz)
- Containers (Carol Willing)
- Resourcing and planning (Kostas Kavoussanakis)

10:00-10:20 Break
10:20-11:50 Break-out discussion & writing sessions - 2
11:50-12:30 Reporting back from breakout sessions - 2
12:30-13:30 Lunch (Sponsored by SSI)
13:30-14:45 Introductions to the "Legacy & Recovery" topics - 3

- Legacy issues (Jamie Shiers)
- Heritage matters (Johan van der Knijff)
- Cultural sector issues (Jesse de Vos)
- Recovery issues and efforts (Wiel Seuskens)

- Re-use metrics (Stephanie van de Sandt)

14:45-16:15 Break-out discussion & writing sessions - 3

16:15-16:35 Break

16:35-17:15 Reporting back from breakout sessions - 3

17:15-17:30 Wrap up and close Day 2

17:30-19:00 Reception, finger foods (Sponsored by Netherlands eScience Centre)

10.3 Day 3 - 26 April 2019

09:00-09:05 Welcome to Day 3

09:05-09:50 Introductions to the "Policy & Quality" topics - 4

- Quality metrics (Carsten Thiel)
- A European Software Sustainability Infrastructure (Neil Chue Hong)
- The European Open Science Cloud (EOSC) landscape (Peter Doorn)

09:50-11:20 Break-out discussion & writing sessions - 4

11:20-11:40 Break

11:40-12:10 Reporting back from breakout sessions - 4

12:10-13:10 Lunch break (Sponsored by SSI)

13:10-14:10 Panel discussion on areas of future work in software development, legacy & policy. Panelists - Neil Chue Hong, Carol Willing, Catherine Jones, Daniel S. Katz and Patrick Aerts, Chair - Shoaib Sufi

14:10-14:30 Workshop Report next steps (Shoaib Sufi)

14:30-14:50 The WOSSS (Patrick Aerts)

14:50-15:00 Wrap up and close WOSSS19

11 References

1. Crouch S, Hong NC, Hettrick S, Jackson M, Pawlik A, Sufi S, et al. The Software Sustainability Institute: Changing Research Software Attitudes and Practices. Computing in Science & Engineering [Internet]. 2013 Nov [cited 2018 Nov 12];15(6):74–80. Available from: http://ieeexplore.ieee.org/document/6731384/

2. Aerts PJC. SUSTAINABLE SOFTWARE SUSTAINABILITY - WORKSHOP REPORT [Internet]. [cited 2019 Jan 9]. Available from: https://doi.org/10.17026/dans-xfe-rn2w

3. Hettrick S. Research Software Sustainability - Report on a Knowledge Exchange Workshop [Internet]. Jisc; 2016 [cited 2019 Nov 18]. Available from: http://www.knowledge-exchange.info/event/software-sustainability

4. de Souza MR, Haines R, Vigo M, Jay C. What Makes Research Software Sustainable? An Interview Study With Research Software Engineers. arXiv:190306039 [cs] [Internet]. 2019 Mar 14 [cited 2019 Nov 25]; Available from: http://arxiv.org/abs/1903.06039

5. Wilkinson MD, Dumontier M, Aalbersberg IjJ, Appleton G, Axton M, Baak A, et al. The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data [Internet]. 2016 Mar 15 [cited 2018 Nov 23];3:160018. Available from: https://www.nature.com/articles/sdata201618

6. Paula Andrea Martinez, Christopher Erdmann, Natasha Simons, Reid Otsuji, Stephanie Labou, Ryan Johnson, et al. Top 10 FAIR Data & Software Things [Internet]. 2019 Feb 1 [cited 2020 May 21]. Available from: https://zenodo.org/record/3409968

7. Lamprecht A-L, Garcia L, Kuzak M, Martinez C, Arcila R, Martin Del Pico E, et al. Towards FAIR principles for research software. Data Science [Internet]. 2019 Jan 1 [cited 2019 Nov 18];Preprint(Preprint):1–23. Available from: https://content.iospress.com/articles/data-science/ds190026

8. Catalogs and Indices for Finding (Scientific) Software [Internet]. Daniel S. Katz's blog. 2015 [cited 2020 May 26]. Available from: https://danielskatzblog.wordpress.com/2015/02/23/catalogs-and-indices-for-finding-scientific-software/

9. Elsevier. Original Software Publications [Internet]. [cited 2020 May 27]. Available from: https://www.elsevier.com/authors/author-resources/research-elements/software-articles/original-software-publications#overview

10. Collins S, Genova F, Harrower N, Hodson S, Jones S, Laaksonen L, et al. Turning FAIR into reality [Internet]. European Commission - European Commission. 2018 [cited 2018 Nov 23]. Available from: https://ec.europa.eu/info/publications/turning-fair-reality_en

11. Sufi S. Software Management Plans in Research Projects [Internet]. Exascale Computing Project. [cited 2020 May 27]. Available from: https://www.exascaleproject.org/event/smp-rp/

12. Wellcome Trust. How to complete an outputs management plan [Internet]. Wellcome. [cited 2020 May 27]. Available from: https://wellcome.ac.uk/grant-funding/guidance/how-complete-outputs-management-plan

13. Smith AM, Katz DS, Niemeyer KE. Software citation principles. PeerJ Comput Sci [Internet]. 2016 Sep 19 [cited 2018 Jul 31];2:e86. Available from: https://peerj.com/articles/cs-86

14. DataCite Metadata Working Group. DataCite Metadata Schema Documentation for the Publication and Citation of Research Data v4.1. 2017 [cited 2020 May 27];72 pages. Available from: https://schema.datacite.org/meta/kernel-4.1/index.html

15. Katz DS, Bouquin D, Hong NPC, Hausman J, Jones C, Chivvis D, et al. Software Citation Implementation Challenges. arXiv:190508674 [cs] [Internet]. 2019 May 21 [cited 2020 May 27]; Available from: http://arxiv.org/abs/1905.08674

16. Chue Hong NP, Allen A, Gonzalez-Beltran A, de Waard A, Smith AM, Robinson C, et al. Software Citation Checklist for Authors [Internet]. Zenodo; 2019 Oct [cited 2020 May 27]. Available from: https://zenodo.org/record/3479199

17. Chue Hong NP, Allen A, Gonzalez-Beltran, de Waard A, Smith AM, Robinson C, et al. Software Citation Checklist for Developers [Internet]. Zenodo; 2019 Oct [cited 2020 May 27]. Available from: https://zenodo.org/record/3482769

18. Hicks D, Wouters P, Waltman L, de Rijcke S, Rafols I. Bibliometrics: The Leiden Manifesto for research metrics. Nature News [Internet]. 2015 Apr 23 [cited 2020 May 27];520(7548):429. Available from: http://www.nature.com/news/bibliometrics-the-leiden-manifesto-for-research-metrics-1.17351

19. Gomes J, Bagnaschi E, Campos I, David M, Alves L, Martins J, et al. Enabling rootless Linux Containers in multi-user environments: The udocker tool. Computer Physics Communications [Internet]. 2018 Nov 1 [cited 2020 May 27];232:84–97. Available from: http://www.sciencedirect.com/science/article/pii/S0010465518302042

20. Bourque P, Fairley RE, IEEE Computer Society. Guide to the software engineering body of knowledge. 2014.

21. Hong NC. Minimal information for reusable scientific software [Internet]. 2014 [cited 2020 May 27]. Available from: https://figshare.com/articles/Minimal_information_for_reusable_scientific_software/1112528

22. Shiers J. Software Preservation and Legacy issues at LEP. 2019 Apr 29 [cited 2020 May 27]; Available from: https://zenodo.org/record/2653526

23. EOSC-hub. EOSC-hub contribution to the EOSC Rules of Participation Consultation [Internet]. [cited 2020 May 27]. Available from: https://eosc-hub.eu/sites/default/files/EOSC-hub%20input%20to%20RoP%20Consultation.pdf

24. Shepherdson J. CESSDA Software Maturity Levels. 2019 Mar 29 [cited 2020 May 27]; Available from: https://zenodo.org/record/2614050

25. Buddenbohm S, Matoni M, Schmunk S, Thiel C. Quality Assessment for the Sustainable Provision of Software Components and Digital Research Infrastructures for the Arts and Humanities. Bibliothek Forschung und Praxis [Internet]. 2017 Jul 6 [cited 2020 May 27];41(2):231–41. Available from: https://www.degruyter.com/view/journals/bfup/41/2/article-p231.xml

26. Sufi S, Jay C. Raising the status of software in research: A survey-based evaluation of the Software Sustainability Institute Fellowship Programme. F1000Research [Internet]. 2018 Oct 3 [cited 2018 Nov 18];7:1599. Available from: https://f1000research.com/articles/7-1599/v1

Last page, intentionally left blank

Photo by Joel Filipe on Unsplash